Debug Methodology for the McKinley Processor

Don Douglas Josephson, Hewlett-Packard Company
Steve Poehlman, Intel Corporation
Vincent Govan, Hewlett-Packard Company

Introduction

The McKinley processor is the result of a joint design effort between Intel and Hewlett-Packard engineers, and is the second processor implementation of the Itanium™ Processor Family. This paper describes in detail the processes and methodology developed for debugging a complex and challenging microprocessor design. An overview of microprocessor debug techniques utilized by HP and Intel is presented, followed by descriptions of debug techniques employed on McKinley. Two case studies of electrical bugs encountered during design debug are presented to demonstrate some of the processes and techniques described previously, and results are discussed. This paper is a companion paper to the "Test Methodology for the McKinley Processor" paper [1], which describes the processor and processor test methodology in general.

Microprocessor Debug Overview

Processor debug begins once first silicon is received after the initial design is completed. Debug can be separated into two major components – functional verification and electrical characterization. Functional verification consists of verifying that the device operates correctly at a nominal operating point. Electrical characterization expands the window of operation of the processor to make sure that it operates properly at the specified electrical and environmental limits that the processor is specified to operate at. Functional verification and electrical characterization tasks are intimately related, and often techniques developed for functional verification can be applied with great efficacy in electrical characterization.

These two components of debug are performed on two different platforms – high speed automated test equipment (ATE), which is also commonly used for manufacturing test, and in-situ debug in systems that the processor is designed to function in. Each platform offers advantages and disadvantages in the process of debug, but fortunately the platforms are often complementary, and both tester and system debug methods have been used in McKinley processor debug.

Functional Verification

Functional verification is the process of verifying that the design behaves in a logically correct fashion at a nominal operating point, with no regard to frequency, voltage or other such environmental parameters [2]. Testing is done at reduced frequencies or voltages that are "safe".

Functional verification consists of initializing the processor to a known state, executing code sequences on the processor, and comparing architectural internal state (e.g. register results) to simulated values or to values obtained from a previous "architectural reference" (i.e. a previous processor system) to determine if the processor is operating correctly. Two different methods of testing are used.

Focused cases are written by hand in assembly language to cover various features of the design. This is most effective when limited in scope to small regions of the chip, and these cases are often leveraged from the initial pre-silicon verification and simulation work. Designers intimately familiar with blocks on the chip often write these cases to test areas of functionality that are the most "risky" or hard to test via other methods.

The other method, which is often the most effective method of finding functional bugs, is to use a code generation process to generate instructions and data in a pseudo-random fashion. These are then simulated in an architectural simulator (not the RTL model on which the chip design was based). The same instruction and data sequences are applied to the processor under test, and the architectural state from the simulator and the processor are compared to ensure that they match. Small test cases of 100's of instructions are run on the processor with very high throughput using this technique. Functional bugs can rapidly be identified with only hours or days of testing in systems by analyzing failure signatures. Several different types of random code generators may be employed when testing a design, with each one focused on different functionality within the processor. Random code generators are also highly configurable, allowing the debugger to control the probability that certain operations will be picked over others.

Electrical Characterization

Electrical characterization expands functional characterization by attempting to "open the window" of the operating region of the processor [3]. Commonly, environmental conditions like voltage and temperature are varied to extremes of the specified operating region for the processor, and operation of the processor is verified at these points.

Another important factor in how the design behaves is process variation. Since all silicon processes exhibit some natural variation in processing, it is important to verify that design will still operate correctly if such variations occur in the future when the part is manufactured. This is achieved through the fabrication of "skew" wafer lots, where manufacturing process parameters are varied to change the electrical performance of the processor in a controlled fashion to the extremes of expected process variation.

In order to explore design margin problems, fast and slow transistors are produced by artificially shortening or lengthening gate lengths, and interconnect parameters (e.g. line widths and spaces between lines on the same layers) are also varied. Such controlled experiments allow design marginalities to be explored in a deterministic way early on in the debug process, instead of waiting for natural process variation to possibly expose design margin problems.

Using the techniques of functional verification described earlier, the processor is tested at extremes of temperature, voltage, and frequency, using parts from different skew lots, in a systematic way, in both testers and systems. This process is designed to expose design marginalities as early in the debug process as possible so that they may be identified and fixed in later revisions of the design.

While focused, hand written cases can be very effective in finding circuit margin problems, random code generators are often the tool of choice to expose margin problems. With the vast number of possible combinations of data between wires on the chip which are often completely unrelated to one another, massive amounts of targeted random testing in many systems is the most effective way to find unexpected design errors, as such problems often elude the best efforts of designers and pre-silicon verification tools to find them.

The concept of "shmoo to failure" is employed in electrical characterization. What this means is that instead of simply verifying that the processor operates in a defined region as specified for customers, the debug team actually pushes outside the specified operating envelope, and attempts to understand the underlying causes of failures outside of it. While this takes extra effort and time, it is usually well worth it, as sometimes design marginalities "outside the envelope" of operation can suddenly move "into the envelope" once the worst-case conditions for the failure are discovered. This leads to a very high quality, robust design that has been well tested even outside its specified limitations.

Tester Debug

Debugging on a tester is the traditional approach to silicon bring-up and debug. The tester environment provides an environment that is not dependent on system limitations. The tester environment allows the designer to vary voltage, frequency, and temperature without affecting other system components. The designer has the freedom to adjust timings and voltage swings on a per-pin basis. Device inputs and outputs are determined by a pattern that is generated by a simulator, or even hand-modified by the designer. These features provide an extremely flexible environment that can be used to characterize the device and debug performance issues. For example, frequency and voltage can be varied to create shmoo plots [4] of device performance and can be used to help diagnose design marginalities [5]. Additionally, access to test features in the design like scan and clock manipulation allows for debug of device failures. Testers can also loop on patterns that pass/fail to allow use of analytical equipment such as parametric analyzers, oscilloscopes, electron beam and laser voltage probes [6] in debugging.

System Debug

Debugging the processor in systems is quite different than debugging it on a tester. Obviously, there is far less control over bus timing and electrical parameters for the processor bus than on a tester, where even individual pins can be set to different timings and levels. However, since comparatively little of the circuitry of the processor is involved in external bus operations, a great deal of internal characterization can be performed in systems as long as voltage, frequency and temperature can be controlled.

System vs. Tester – Advantages and Disadvantages

Processor debug work in systems offers several advantages over debug work on testers. Systems are far cheaper than high-speed testers. A large number of systems can be built for the price of one high performance tester. Besides economics, this also makes sense from a

throughput issue – many more parts can be characterized more rapidly since many more systems are available. This essentially parallelizes the work of processor characterization.

Systems are also far more reliable than testers, and even if one is "down", many more are available, whereas the loss of one tester out of only a few can be a serious impediment to debug progress. Finally, one of the most important benefits in using systems is that many more vectors can be applied to test the device in a system as compared to the tester. Cycle-accurate simulation is necessary to obtain the logical values to be applied/observed at the pins of the device on a tester, and tester memory is often limited in depth and slow to reload. In contrast, a system is already designed to be able to apply the correct "vectors" to the external bus interface, and "vector depth" is only limited by available memory in the system and creativity in applying "vectors" to the device.

There are a few disadvantages to processor debug in systems. One is the obvious lack of fine control of electrical parameters like voltage levels, timing specs and loading of the bus. This makes full bus characterization in systems more difficult than on a tester, and this is a task that is very well suited to tester characterization.

Another problem is that operation of the entire system at voltage or temperature extremes can cause problems unrelated to the processor to exhibit themselves in other system components. For example, if a clock generator chip supplying the processor exhibits undesirable behavior at high voltage due to problems within the clock generator chip itself, it can be difficult to determine that it is affecting processor operation, and debuggers can be led down blind alleys while debugging. Therefore, it is very important to attempt to isolate the processor environmental factors such as voltage and temperature from the rest of the system as much as possible. For the McKinley processor debug system, care was taken to design the system to allow the environment of each individual processor to be varied without affecting other system components. For instance, chip-level temperature controllers are used to control the processor temperature, and processors have their own power pods that can be programmed independently of each other.

Finally, use of scan features to assist debug is often more complicated in systems than on a tester. Tester behavior is (by nature) very deterministic and internal state of the design is thus well controlled and always in synchronization with the tester. However, system events (such as memory refresh or interrupts) may cause repeatability problems in debugging and certainly will cause differences in internal chip state from run to run in a system shmoo. This can greatly complicate the identification of failures and comparison of "good vs. bad" scan signatures in systems as compared to the tester environment.

McKinley Debug Feature Overview

The processes described above are used to expose problems in the design for further investigation. To enable this, numerous features are present in the McKinley processor to assist in the debug of functional errors and electrical marginalities present in the design. These include a sophisticated debug unit which can monitor processor operation and provide triggers to other test and debug functionality, a probe mode feature which allows access to internal processor state, the capability to do several different kinds of clock manipulation (including internal analog and digital edge manipulation), and traditional scan features to allow internal state to be observed while the chip operates and when it is halted. Some new capabilities are also described, including programmable clock skewing, access to the IEEE 1149.1 port via processor instruction streams, and a feature to extract timing information for some signals within the design called "timing on the fly".

Debug Unit

The McKinley processor has a dedicated debug unit providing very flexible trigger mechanisms that can be programmed to respond to a wide variety of events. The trigger responses are set up through a series of control registers used to identify the event to respond to and the response. These registers are accessible both via processor specific instructions as well as through scan via the processor's Integrated Test Controller (ITC), which controls test operations in the device. Trigger events can be generated from external pins, instruction bundles, addresses is various stages of the instruction and data pipelines, internal counter overflows, performance events and other triggers. By nesting trigger events and combining them with performance counters, complex failure modes and events can be recognized and various responses can be initiated. Responses to the triggers can be any one or a combination of the following: stop or digitally manipulate the clock, perform an analog manipulation of the clock, perform a "sample/timing on the fly" operation, enter probe mode to allow external access to device state, access control registers (write/read) and toggling of external debug pins to enable bus data capture.

Probe Mode

Probe mode was first implemented on Intel Pentium® processors [7]. Prior to the existence of probe mode, ICE emulators (in-circuit emulators) were used to debug systems. Probe mode allows the customer to debug systems by allowing them to "probe" the internal state of the processor. The user can halt execution by going into a special microarchitectural stall in response to a trigger event. The user is able to enter this mode on any instruction boundary by stopping the fetching of additional instructions. This allows the remaining instructions in the pipeline to complete, and any pending memory operations to finish. At this point the processor stalls.

Examination of internal state is controlled via a personal computer connected to the IEEE 1149.1 port of the processor. The software on the PC provides the user interface to define what data is to be collected and displayed. Processor instruction sequences can be executed using a scan chain to allow the user to examine/modify internal state or external memory; essentially, normal processor instructions are scanned in and executed in a single step fashion. The interface allows the user to restart the processor by entering a special instruction to return to normal operation. This feature is invaluable for silicon debug in systems as it allows excellent access to all memory and registers with an easy to use interface.

Clock Manipulation

As core clock periods shrink, precise on-chip clock control is a requirement for product debug and testing; external clock manipulation is no longer adequate for the debugging of high performance processors [8]. McKinley provides numerous methods through scan/code programmable debug registers and trigger responses to perform both digital (skip or extend clock phases) or analog (expand or shrink by picosecond increments) manipulation of the main clock (denoted as SLCBO) in the design. Clock manipulation is a key feature for enabling the debug of complex electrical problems encountered in high performance designs.

An 8-bit control register configures the source of the main clock for the processor. The possible sources are the normal on-chip PLL, an external bypass clock source, an internal ring oscillator and an analog edge manipulator. The edge manipulator in turn can use either the PLL or the bypass clock as an input, and thus modify their edges as desired. Once the control register is loaded via the ITC, the CLU switches over to the selected clock source.

Digital Manipulation

The digital manipulation of the clock is controlled cycle-by-cycle in the ITC based on debug triggers provided by the on-board debug block. Refer to Figure 1 for an example of digital clock manipulation. The "halt_high" and "halt_low" signals are used by the clock unit (CLU) to alter the clock waveform. If "halt_high" is high, the CLU will not allow a falling edge. If the "halt_low" signal is high, the CLU will not allow a rising edge. The halt signals are configured in a 96 bit clock command register in the ITC. Bits 47:0 are the halt_low register and 95:48 are the halt_high register. Logically, these are shift registers.

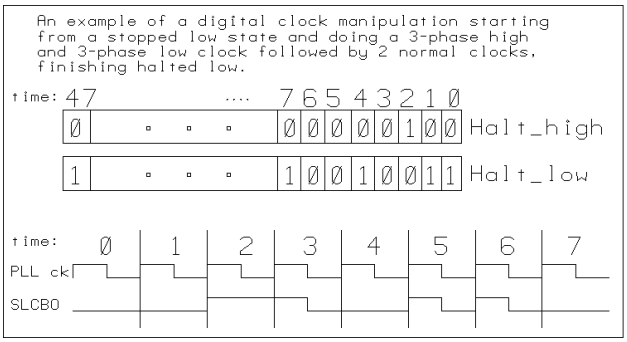

Figure 1 – Digital Clock Manipulation

During each clock cycle after a debug trigger, bit 0 of both the halt_high and halt_low registers is shifted to the CLU as the register is shifted, which causes the internal CPU clock to be modified based on setting of the bit pattern in the register. This allows very flexible manipulation of the internal clock waveform over a 48-cycle window beginning with the debug trigger. The digital clock manipulation can be terminated with the clock stopped low or high, or with the clock free running. Such manipulations can be performed on a tester or in a system. Although processor operation cannot be resumed after a clock halt in a system without a reset, digital manipulations can still be performed.

Analog Manipulation

Analog manipulation of the clock (moving the rising and/or falling clock edges by a programmable fixed delay) is accessed through the ITC and executed in the CLU via a debug block trigger or an ITC command. This feature is used to artificially shrink/lengthen clock phases or cycles to assist in the debugging of the processor. The ITC is

used to change the clock source to an analog edge manipulator, and also defines the desired manipulation and the duration of the manipulation. An 11-bit register defines the analog nature of the manipulation. Fine and coarse delay settings are programmable in the register for both the rising and falling edges of the clock signal. Edge movement of up to 120 ps under typical conditions is possible, with resolution of approximately 5 ps.

The register used for digital manipulation is also used here to control when clock phases start being affected by the analog manipulation and when the manipulation stops. After a debug trigger, the register counts clock cycles and asserts a reset signal to the CLU to stop the analog manipulation. Edges are moved by the selected delay amount based on the programming of the analog manipulation delay register. Typically, only several edges programmed in the clock command register are affected; when analog manipulation is required for longer than 48 cycles, an "infinite manipulation" mode also exists to change either all rising or falling edges, or both.

Programmable Clock Skewing

Localized clock buffer skewing capability on the McKinley processor allows for clock tuning on a regional basis. The individual clock regions on the device can be skewed relative to the other clock regions on the processor. This allows the processor to adjust for clock routing issues and other possible contributions to clock skew between regions. For debug, the global clock route can be tuned to allow for higher performance or to assist in debugging of speed paths or races between circuits controlled by different clock buffers. By adjusting the clock skew at driver or receiving block, interblock speed paths or other timing-related problems can be isolated. This feature can also be useful in sourcing design marginalities through systematic adjustment of each clock region relative to the others. The clocks are skewed by setting values into a scanpath that controls the clock buffer delays. The skewing operation is controlled by another scanlatch that determines if default powerup values or scanned-in values are used. The clocks can be skewed relative to each other by approximately +/- 70 ps.

Scan Debug Techniques

Due to the nearly full scan methodology used on the McKinley processor, excellent visibility into the internal operation of the device is available on the tester and in the system. Two scan-based techniques are available to assist in debugging the processor during functional and electrical validation. Sample on the fly [7 – pg. 298, 9] is an important capability that gives a limited overview of the internal state of the processor while it continues to operate. In combination with the debug block, it allows the examination of approximately 24,000 internal nodes of the processor while not disturbing normal operation. Using this capability, it is possible to capture a snapshot of device operation on a tester or in a system. Since this mode does not affect device operation, it is not necessary to stop the clock to scan data out of the chip; this can be done during operation after a sample operation has been completed. This allows rapid, multiple samples to be captured in a system environment.

Once a failure has been narrowed down to a narrow range of clock cycles, it is possible to stop the clock in the device using a debug trigger on a specified clock cycle and "scandump" the contents of all the 36 regular scanpaths for very fine-grained visibility into failures. This is a very useful technique in debugging circuit failures as approximately 136,000 state elements can be examined. However, once the clock has been stopped, the processor cannot be restarted from the same point in a system, so performing this over and over is extremely inefficient, especially for debugging failures in code which takes minutes to get to a failing point (for example in OS debug or in random code verification programs). For this reason, sample on the fly is typically used to narrow a failure down to a small region in a system, and then scandumps are taken to make a final determination of what the failure is. Both modes are used on testers.

Timing on the Fly

A new capability called "timing on the fly" implemented in the McKinley processor allows the timing of selected internal signals to be determined with an accuracy of approximately 60 ps. Whereas sample on the fly only allows determination of logical signal values during a given clock cycle, timing on the fly adds the capability to capture some timing information as well. As decreasing feature sizes make techniques like e-beam and laser voltage probing more difficult, this is an alternative way to capture timing information internal to the processor. This technique is similar to sample on the fly, but utilizes a different and more aggressive clocking scheme. In order to capture timing information, narrow pulses are used to capture signals in scanlatches instead of phase clocks (Figure 2).

These pulses are created using a small (45 x 22 μm), local delay locked loop (DLL), which is designed to be insensitive to process, voltage and temperature variation. The delay generated by the DLL is programmable via scanlatches that are set during the initialization for sample

on the fly. At this point, the DLL adjusts itself to provide a precisely delayed clock locked to the local regional clock. Once the sampling trigger fires, the delayed clock is used to capture numerous local signals of interest into scanlatches using local pulse generators on the cycle desired.

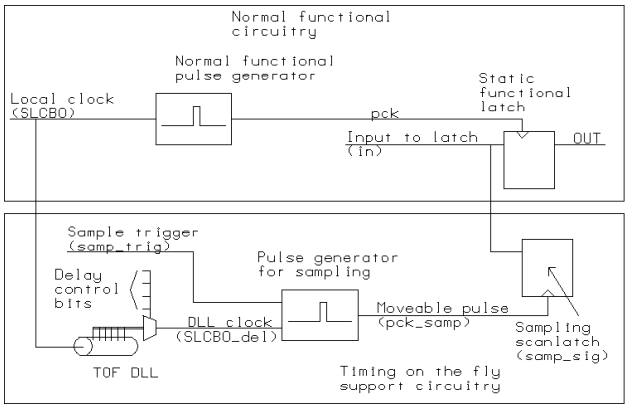

Figure 2 – Timing on the Fly

By programming the DLL to provide different delays, the sampling pulse can be "shmooed" across a clock cycle in order to capture signals at different points within one clock cycle. By examining the logical value captured at the different delay points, it is possible to determine with high accuracy when a signal transition actually occurred. The output clock of the DLL is adjustable from approximately –60% to +20% of the local clock rising edge to allow capture of signals considerably before and slightly after the local pck clock pulses used by the functional latches in the design.

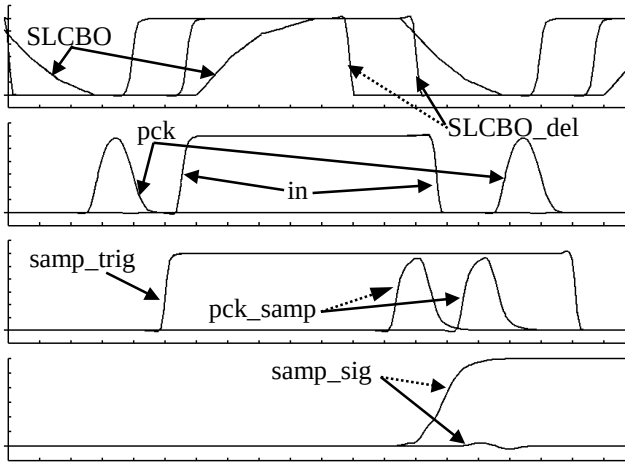

Figure 3 – Example Waveforms

An example of the timing used is shown in Figure 3. Time is on the X-axis, and voltage is on the Y-axis. SLCBO is the main clock, which is fed to the input of the DLL. The DLL then outputs a delayed clock based on the delay control bits. In this example, two different delays are shown for the relevant signals. The effect of an earlier delay is shown with dashed arrows, and a later delay with solid arrows, for the SLCBO_del, pck_samp, and samp_sig waveforms.

The falling edge of SLCBO_del generates the pck_samp pulse, which captures the current value of the signal in into the sampling scanlatch (represented by the samp_sig waveform). As shown, when the earlier delay is used (dashed arrows), the high value on the signal in is captured by the earlier version of the pck_samp pulse, and this is shown by samp_sig rising. When the later delay is used (solid arrows), samp_sig remains at a low value, since the signal in has transitioned low by the time the second, more delayed pck_samp pulse occurs. Finer time delays are possible, but were omitted for clarity.

This capability can be very useful for determining the amount of timing margin in critical paths, and in other areas where understanding timing margins can lead to insight into circuit failures. Approximately 1,000 signals in the device are captured using this technique. Signals were selected based on timing criticality during design based on timing analysis and also based on designer insight in areas where aggressive and new circuit design techniques were used.

This capability has not yet been evaluated in the design. Some drawbacks include the necessity of careful design of the clocking networks that distribute the sampling pulses (much more critical than for sample on the fly), the additional area overhead required, the complexity of controlling the triggers and programming delays to capture information at the desired times, and the difficulty in verifying proper operation (e.g. the timing of the delays from the DLL is difficult to measure). Finally, only a limited set of signals can be sampled and they must be chosen during the design phase; however, this could be addressed in future designs by providing generic "sampling zones" throughout the chip and multiplexing many more signals into the sampling latches, at the cost of increased complexity in design and determining actual timing of signals passed through additional extraneous logic.

IEEE 1149.1 Port Access via Code

Another new technique implemented in the processor allows access to the normal IEEE 1149.1 port and the ITC through execution of normal code sequences on the device (Figure 4). To implement this feature, two 64-bit registers

are used to replace the normal TDI, TMS and TDO pins. Both registers are accessible using processor instructions. One register provides TDI values and captures TDO values, while the other provides TMS values. In this special mode, TCK is derived by dividing the main processor clock by 8. Note that TCK is derived from the processor clock only in this particular mode; during normal operation of boundary scan, TCK is supplied from the normal TCK input pin in compliance with the 1149.1 standard. Powering up the processor, pulling the TRST pin, or entering the Test-Logic-Reset state will always cause the normal 1149.1 pins to be selected for communication with the ITC.

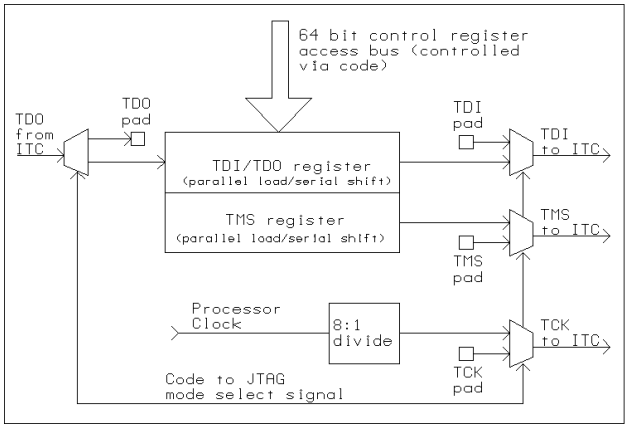

Figure 4 – Code Access to the ITC

In order to use this feature, first both registers are programmed with 64 bits of the desired TDI and TMS data in order by loading the data into the registers in parallel. Once these are programmed, a control register that contains the number of TCK clocks and an enable bit is programmed. At this point, a multiplexer in the 1149.1 pins is activated to select the registers as a source instead of the normal 1149.1 pins, and the two registers are configured to act as serial shift registers, providing one bit of data to each of the TDI and TMS pins using the TCK generated from the divided processor clock. At the same time that the TDI data is shifted out of the register, TDO data is then captured back into the same register. Once the 64 bits of data have been applied, the TCK is stopped and the TDO values are read (in parallel) out of the register, and new values to be applied can be placed in the registers to apply more vectors to the design.

This capability allows access to any test features normally accessible via the standard 1149.1 port. This is useful for debugging in systems to access scan-based test and debug features without having to include considerable additional hardware to duplicate access via normal methods, which reduces wiring and area requirements. It can also be useful for allowing access to many different test features in systems in the field for debug or diagnostic purposes. For example, a program can be downloaded into the computer and run to help diagnose the processor using the internal test features, without requiring any additional debug hardware to be attached to the device.

One major drawback with this mode is that the processor must be functional enough to execute the required code sequences to read/write the TDO/TDI and TMS registers at the operating point where chip debug is being done, which may not always be possible. Another problem is that the normal scanpaths in the device cannot be accessed in this mode, as scanning them requires stopping the chip clock, which would obviously stop this mode from working. In these events, the normal 1149.1 hardware interface port can be used.

Debug Results

All processor designs have bugs, and McKinley is no exception. Roughly 20 functional bugs have been identified in the design currently; however, only a few limited functional verification in any way. Debug of functional design errors is usually easier than diagnosis of electrical failures. Typically functional failures are identified through code experiments and RTL simulation, supplemented by scan data collected while running test code on the design, and time to root cause is usually very quick (hours to a few days).

Electrical characterization mainly uses code experiments, probe mode, scan debug techniques and clock manipulation to discover the root cause of problems. Analog clock manipulation is particularly useful in identifying “sensitive” clock phases/cycles in a test case, which can then be explored with scan tools. In addition, clock manipulation can often help in duplicating a previously found failure by pinpointing the exact cycle of failure; at this point scan data can be taken at the identified cycle and compared to scan signatures for known bugs. Once identified, bugs can sometimes be avoided by using clock skewing.

Shown in Table 1 is a breakout by percentage of some different electrical bug types found on McKinley. Typical time to find the root cause and a fix for an electrical bug is less than two weeks. In order to illustrate some of the electrical failure types encountered in processor designs, two different electrical bugs encountered during McKinley design debug are presented next.

Electrical Bug Type	Percentage of Total
Low Voltage Wall	13%
High Voltage Wall	11%
Speedpath	33%
Test-related	8%
Other	35%

Table 1 - McKinley Electrical Bug Percentages

Clock skew – failure to set data register

This was the very first bug encountered in silicon debug on McKinley. The behavior of the chip in a system and on a tester was such that it would fail to properly operate during initial bus transactions immediately after reset at the voltage the processor was designed to operate at. However, at voltages above and below a band around the design voltage, the part would correctly operate. This of course could have been a major limiter in characterization the design, and greatly concerned the design and debug teams, as numerous parts exhibited the failure. Investigation into why the design did not operate at the nominal operating point was begun immediately.

During the investigation, higher temperatures were found to make the failure go away. Scan data was also collected during the period immediately after reset on a tester. This data indicated that bus fetches were making it into a queue of latches in the bus logic in the processor, but failed to emerge at the other end, which was key in pinpointing the failure. Further investigation of the circuits in the queue pointed to a register with suspect clocking.

Two pulse generators at opposite ends of the clock wire clocked the register in question (i.e. the pulse generators were shorted together by design at opposite ends of the clock line); the generators received differently routed versions of the main processor clock as inputs. It was theorized that if excessive clock skew existed between the two pulse generators for some reason, the voltage/temperature behavior observed could occur. The scan data showed that bits in the middle of the register were the first to fail, which supported the theory. SPICE experiments also showed that if the pulse generator inputs were skewed, the behavior matched that seen in silicon.

At this point, with strong data indicating a clock skew problem, laser voltage probing (LVP) was performed, which showed that indeed there was a 120 ps difference in the inputs to the generators. At the same time, a hand check of the clock network artwork was performed which found that an earlier point in the network had inadvertently been contacted by an autocontacter tool, which resulted in the additional clock skew observed in probing.

Once the problem was found, a simple FIB (focused ion beam) edit was performed to cut this extraneous input to one of the generators (the generator was also already connected to the network at the proper delay point) and restore the designed-in delay; this showed that the edited part worked correctly. At this point, a simple mask change was made to eliminate the two extraneous contacts permanently. With the fix, the chip was found to operate well across the voltage range desired for initial characterization. Scan debug hooks and LVP were key in identifying the failure. Initial observation to root cause diagnosis and a FIB fix was 8 days.

The problem was missed in design because the clock network was designed, laid out and simulated in isolation from the final chip artwork, using assumptions for underlying metal, and a SPICE simulation from a full RC extract was not performed. After identifying the root cause of the problem, an extraction/simulation process was performed from all the artwork for the design to be sure there were no additional errors in the clock network, and none were found. This bug is indicative of how extremely minor mistakes (two inadvertent additional contacts added by a script in artwork) can cause serious problems in a design, and indicate the exceptional care that must be taken in design to ensure proper operation.

Low voltage wall – failure to set latches

This bug was encountered early on in design characterization. The failure observed was a low voltage wall in a shmoo of randomly generated test cases, where the design failed to operate at any frequency below a certain low voltage. Through the use of probe mode and code experiments in a system, the failure was shown to be due to a certain instruction being performed incorrectly. Additional shmoos were done by varying temperature and using faster and slower “skewed” parts to determine circuit sensitivities.

By performing rigorous code experiments on testers and in systems, coupled with circuit analysis, the failure was narrowed down to several latches related to the instruction that caused incorrect operation at low voltage. SPICE

investigation of the questionable circuits showed that the inverter driving the inputs to four large latches was unable to clear the latches at lower voltages. This was the result of a long, resistive metal route from the inverter to the latches, the inverter being inadequately sized, and the fact that all four latches were set from the single line in this case. The PFET feedback in all four latches (since they were latches with large drive capability) was relatively large, which overwhelmed the capability of the inverter to pull the internal latch nodes down through the input pass gates to the latches when they were updated simultaneously.

Scan data taken showed that the four latches indeed failed to set as hypothesized. To fix the design quickly, the metal line to the latches was made less resistive, which allowed the inverter to overdrive the latches properly. To avoid this bug until revised silicon was available, random code generators were modified to avoid the failing instruction in electrical characterization. Code experiments, probe mode, and scandumps were key in identifying the failing circuit, and the bug took about 16 days to root cause.

Results

The debug trigger, clock manipulation, probe mode and scan debug features described worked well in first silicon and were essential for early debug of the processor. Probe mode had a couple of minor bugs that required trivial workarounds to avoid. 1149.1 access via code has been tested and works well functionally. This will be used again in future designs to greatly speed up access to scan information in systems without having to use very slow (10 MHz) 1149.1 hardware. Timing on the fly has not yet been evaluated due to the difficulties in verifying proper operation of this feature.

Some difficulties were encountered with digital clock manipulation (which were expected, but could not be easily avoided without significant effort in the design). Since McKinley uses a synchronous bus, the internal core clock is a multiple of the bus clock, and removal of single core CPU clocks can force the CPU core to be out of synchronization with the bus. This can be avoided by always halting for the same number of core CPU clocks as the current bus multiple. However, this only works for cases where a bus transaction is not occurring at the time the core clocks are "skipped" (since the CPU core will not receive the bus transaction since it is not being clocked during the transaction). Due to the large caches used in McKinley however, it is usually possible to force test cases to be entirely cache-resident for clock manipulation. This issue can also be completely avoided when using an asynchronous bus protocol (where the CPU and bus are in different domains), which we have successfully proven in other designs.

A race at very low voltages was found when using debug triggers for clock manipulation that sometimes caused the clock edge being manipulated to be off by one cycle. This was the result of inadequate verification between the debug trigger clock domain and the clock unit clock domain. The bug was easily fixed in the design early in debug and has not been a limiter in debug. Analog manipulation of the clock has worked very well and has been a primary debug method for determining what clock cycle an electrical bug is occurring on. The clock skewing features have also been found to be an excellent method of temporarily "fixing" electrical bugs that are dependent on clock skew (such as speedpaths and races) between blocks, by decreasing the clock skew between them. This can greatly accelerate debug as these bugs can block further debug progress unless they can be avoided. Clock skewing has also been a good method of sourcing additional design sensitivities when performed in a systematic way on testers and in the system. Increasing the clock skew between two blocks can "poke" the design and stimulate new failures and sensitivities that might not have been found until much later due to process variation.

Debug of design marginalities indicates that the debug features implemented are extremely effective in analyzing and root-causing circuit failures much more rapidly than in past designs (days instead of weeks). Several complex electrical marginalities were root-caused and fixed with FIB edits within days of observing the initial failure mode during characterization efforts. These would have taken much longer to identify using our previous debug methodologies. Reducing time to root cause for bugs has a very beneficial effect on time to market, and is a key reason to include sufficient debug hardware in a design.

There are several areas for future investigation with regard to debug enhancements. A way to identify and root cause duplicate bugs more quickly is desirable, as running random code in systems can rapidly hit the same electrical bug in the design in different ways (i.e. the same bug manifests itself in multiple ways in different code sequences). However, since all anomalous failures must be investigated to determine the root cause of the failure, we have sometimes spent a large amount of time in debug proving that some bugs were simply duplicates of ones we had already found and root caused. To date, we have found roughly 50 duplicate electrical bugs.

New methods to improve our ability to debug processors in systems by capturing more information more quickly, over many cycles of operation will be investigated. These

could improve our ability to investigate timing issues in the design. While these methods could reduce dependence on high-speed testers for silicon debug, they can also aid in the porting of failing test cases from systems to testers, where they can be analyzed more carefully and in greater detail.

Conclusion

While McKinley processor characterization is ongoing, we are very encouraged by the initial results of the characterization effort. All debug features used to date are functional, with only minimal workarounds required. The time required to debug failures has been greatly reduced from that required in previous designs, largely as a result of the scan debug, probe mode and clock manipulation features present in the design; this was a primary goal in the design of the debug features for McKinley. Finally, several opportunities for improvement in debug capability in future designs have been identified as a result of our work.

Acknowledgements

The McKinley processor debug methodology was a result of a joint effort between Intel and HP engineers, and many dedicated and talented individuals were involved in the debug methodology decisions and implementation. In particular we would like to acknowledge the work of Dan Dixon, Sam Naffziger, Bob Gottlieb, Steve Wells, Eugene Berta, Ferd Anderson, Gerard Blair, Wayne Kever and Jim Finnell for their direct contributions to the debug features in the design. In addition, we acknowledge the exceptional efforts of the debug and design teams in characterizing and improving the design. For management support and leadership during design of the test and debug circuitry, as well as during debug, we are indebted to Joe Butler, Rory McInerney, Tom Meyer, and Dan Swanson.

References

[1] D. Josephson, et al., “Test Methodology for the McKinley Processor”, Proceedings of the International Test Conference, 2001.

[2] S. Mangelsdorf, et al., “Functional Verification of the HP PA8000 Processor”, Hewlett-Packard Journal, August 1997, pg. 29.

[3] J. Bockhaus, “Electrical Verification of the HP PA8000 Processor”, Hewlett-Packard Journal, August 1997, pp. 32-39.

[4] K. Baker, J. V. Beers, “Shmoo Plotting: The Black Art of IC Testing”, IEEE Design and Test of Computers, July/September, 1997, Volume 14, Number 3, pp. 90-97.

[5] W. Huott, et al., “The Attack of the ‘Holey Shmoos’: A Case Study of Advanced DFD and Picosecond Imaging Analysis (PICA)”, Proceedings of the International Test Conference, 1999, pp. 883-891.

[6] M. Paniccia, et al., “Novel Optical Probing Technique for Flip Chip Packaged Microprocessors”, Proceedings of the International Test Conference, 1998, pp. 740-747.

[7] A. Carbine, D. Feltham, “Pentium® Pro Processor Design for Test and Debug”, Proceedings of the International Test Conference, 1997, pp. 298-299.

[8] S. Tam, et al., “Clock Generation and Distribution for the First IA-64 Microprocessor”, IEEE Journal of Solid State Circuits, Volume 35, Issue 11, November 2000, pp. 1545-1552.

[9] D. Josephson, et al., “Test Features of the PA7100LC Processor”, Proceedings of the International Test Conference, 1993, pg. 771.